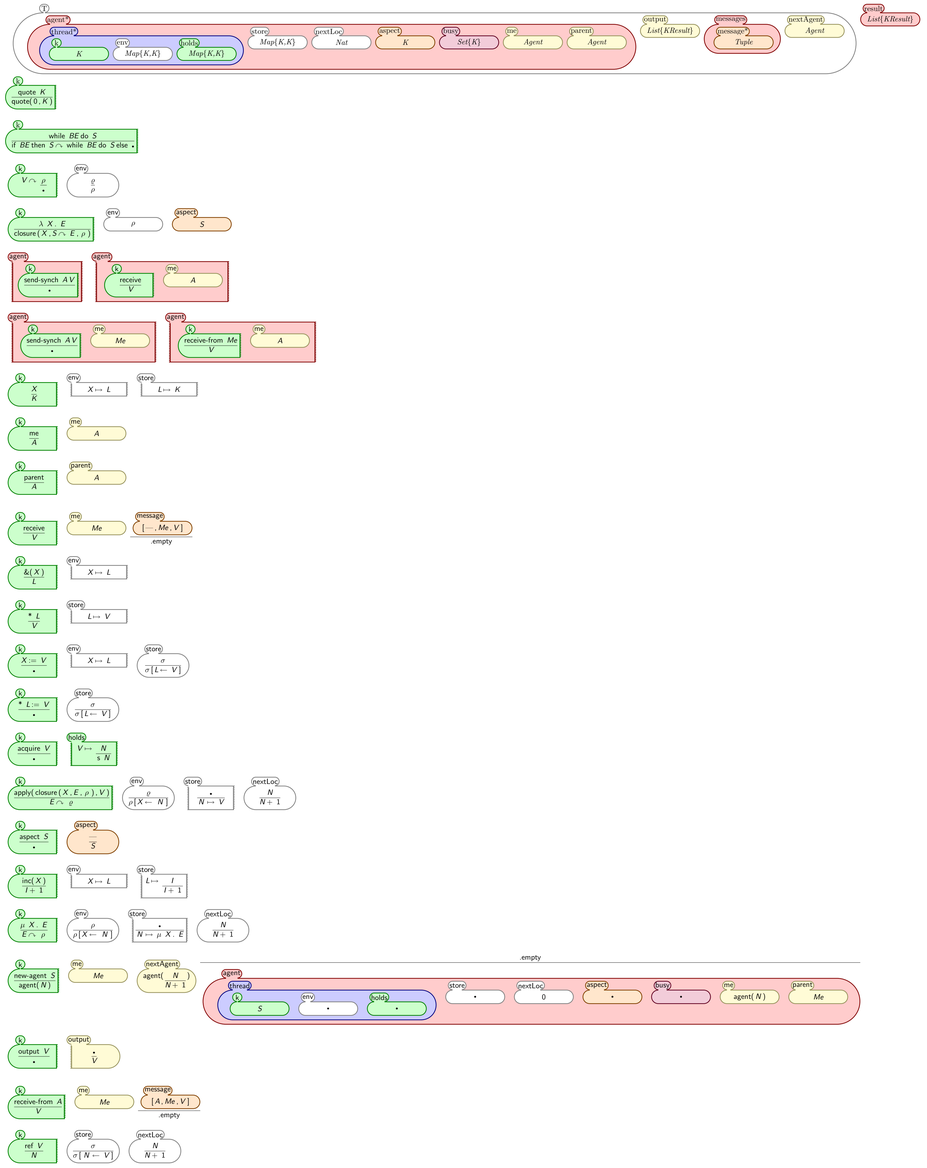
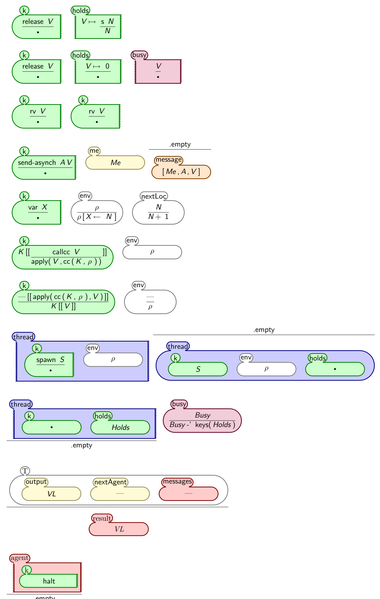
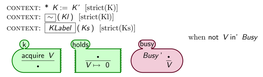
```
Module KCHALLENGE
  imports BOOL-K-SYNTAX+FLOAT-K-SYNTAX+INT-K-SYNTAX
  imports NAME-K-SYNTAX
  imports KMAP{K,K}
  imports TUPLE
  syntactic constructs:
    Exp ::= me
          | parent
          | randomBool
          | receive
          | * Exp [strict]
          | ++ #Name [renameTo inc]
          | callcc Exp [strict]
          | eval Exp [strict]
          | new-agent Stmt
          | not Exp [strict]
          | quote Exp
          | receive-from Exp [strict]
          | ref Exp [strict]
          | unquote Exp
          | &( #Name )
          | Exp Exp [renameTo apply strict]
          | Exp ≤ Exp [seqstrict]
          | Exp * Exp [strict]
          | Exp + Exp [strict]
          | Exp and Exp [strict(1)]
          | Stmt ; Exp [renameTo ↷]
          | λ #Name . Exp
          | μ #Name . Exp
    Stmt ::= halt
           | acquire Exp [strict]
           | aspect Stmt
           | output Exp [strict]
           | release Exp [strict]
           | rv Exp [strict]
           | spawn Stmt
           | var #Name
           | Exp := Exp [strict(2)]
           | Stmt ; Stmt [renameTo ↷]
           | send-asynch Exp Exp [strict]
           | send-synch Exp Exp [strict]
           | while Exp do Stmt
           | if Exp then Stmt else Stmt [strict(1)]
  semantic constructs:
    Agent ::= agent( Nat )
    Config ::= [ Stmt ]
    KProper ::= K ↷ K [strict]
              | K ⊔ K [strict]
              | quote( Nat , List{K} )
    KProperLabel ::= ~
                   | K
    KResultLabel ::= cc
                   | closure
                   | code
    NeK ::= KLabel
          | Map{K,K}
  configuration:
```

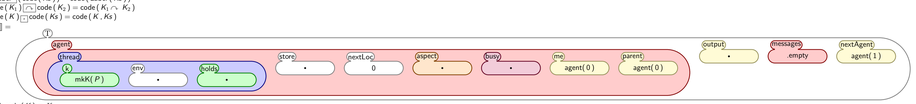




CONTEXT: * K := K' [strict(K)]
CONTEXT: ~ [Kl] [strict(Kl)]
CONTEXT: $KLabel$ (Ks) [strict(Ks)]



~ [code(Kl)] = code(~(Kl))
$Label$ [code(Ks)] = code($Label$ (Ks))
code(K_1) ^ code(K_2) = code(K_1 ^ K_2)
code(K) code(Ks) = code(K , Ks)
[P] =



eval code(K) = K
quote(0 , unquote K) = K
quote(N ,) = code(•)
quote(N , V) = code(V)
quote(N , X) = code(X)
quote(N , K_1 ^ K_2) = quote(N , K_1) ^ quote(N , K_2)
quote(N , K , Ks) = quote(N , K) quote(N , Ks)
quote(N , quote K) = [quote] (quote(s N , K))
quote(N , ~(Kl)) = ~ [quote(N , Kl)]
quote(s N , unquote K) = [unquote] (quote(N , K))
quote(N , $Label$ (Ks)) = $Label$ (quote(N , Ks)) when $Label$ =/= quote and $Label$ =/= unquote

⇒ .empty

F_1 * F_2 ⇒ F_1 * F_2

I_1 * I_2 ⇒ I_1 * I_2

F_1 + F_2 ⇒ F_1 + F_2

I_1 + I_2 ⇒ I_1 + I_2

F_1 ≤ F_2 ⇒ F_1 ≤ F_2

I_1 ≤ I_2 ⇒ I_1 ≤ I_2

false and — ⇒ false

true and BE ⇒ BE

if false then — else S_2 ⇒ S_2

if true then S_1 else — ⇒ S_1

not B ⇒ not B

END MODULE